

# Show Me the Money: Dynamic Recommendations for Revenue Maximization\*

## Full Technical Report

Wei Lu Shanshan Chen Keqian Li Laks V.S. Lakshmanan

Department of Computer Science  
University of British Columbia  
Vancouver, B.C., Canada  
{welu, cshan33, likeqian, laks}@cs.ubc.ca

### ABSTRACT

Recommender Systems (RS) play a vital role in applications such as e-commerce and on-demand content streaming. Research on RS has mainly focused on the *customer perspective*, i.e., accurate prediction of user preferences and maximization of user utilities. As a result, most existing techniques are not explicitly built for *revenue maximization*, the primary business goal of enterprises. In this work, we explore and exploit a novel connection between RS and the profitability of a business. As recommendations can be seen as an information channel between a business and its customers, it is interesting and important to investigate how to make strategic dynamic recommendations leading to maximum possible revenue. To this end, we propose a novel revenue model that takes into account a variety of factors including prices, valuations, saturation effects, and competition amongst products. Under this model, we study the problem of finding revenue-maximizing recommendation strategies over a finite time horizon. We show that this problem is NP-hard, but approximation guarantees can be obtained for a slightly relaxed version, by establishing an elegant connection to matroid theory. Given the prohibitively high complexity of the approximation algorithm, we also design intelligent heuristics for the original problem. Finally, we conduct extensive experiments on two real and synthetic datasets and demonstrate the efficiency, scalability, and effectiveness of our algorithms, and that they significantly outperform several intuitive baselines.

### 1. INTRODUCTION

Fueled by online applications such as e-commerce (e.g., Amazon.com) and on-demand content streaming (e.g., Netflix), Recommender Systems (RS) have emerged as a popular paradigm and often as an alternative to search, for enabling customers to quickly discover needed items. Such systems use the feedback (e.g., ratings) received from users on the items they have bought or experienced to build profiles of users and items, which are then used to suggest new items that might be of interest [1]. The key technical problem on which most RS research has focused is rating prediction and rating-based top- $k$  recommendations: given all ratings observed so far, predict the likely ratings users would give unrated products. In practice, the data is huge, with typically hundreds of thousands of users and items, and is sparse, as users rate very few items. Major rating prediction techniques can be categorized into content-based and collaborative filtering (CF). The latter

can be further divided into memory-based and model-based (cf. §2, also [1, 18, 26]). Here we simply note that for top- $k$  recommendations, a RS predicts ratings for all user-item pairs, and for each user, pushes the items with the highest predicted ratings as personalized recommendations.

The above account is a *customer-centric* view of RS, on which most existing research has focused. We also refer to it as *rating-based* recommendation. There is an equally important, complementary *business-centric* view. From a business viewpoint, the goal of a seller running a RS is to maximize profit or revenue. It is thus natural to ask: When a recommender system has choices of promising items but can only recommend up to  $k$  items at a time, how should it make the selections to achieve better revenue? We refer to this as *revenue-driven recommendation*.

Top- $k$  rating-based recommendations are relevant to revenue, but the connection is weak and indirect: by suggesting to each user the  $k$  items with highest predicted rating, the hope is to increase the chances that some of those items are bought, and there ends the connection. Clearly, there exists a gap: rating-based approaches ignores important monetary factors that are vital to users' product adoption decisions, namely the price and whether such price would be considered acceptable to an individual. Naturally there exists inherent uncertainty in users' purchase decisions due to people's innate valuation [15], and thus successful revenue-driven recommendations must be aware of these monetary factors.

There has been some recent work on revenue-driven recommendations [3, 6, 7] (see §2 for more details). All these previous works focus on a static setting: generate a set of recommendations for various users for a given snapshot, corresponding to a specific time. This is limited because e-commerce markets are highly *dynamic* in nature. For example, the price of many products in Amazon has been found to change and fluctuate frequently [2, 4]. Therefore, narrowly focusing on just one snapshot is restrictive. Instead, it is necessary to be strategic about the recommendations rolled out in a short time period. For instance, suppose a product is scheduled to go on sale in the next few days, it would be wise to strategically postpone its recommendation to low-valuation users (those not willing to pay the original price) to the sale date, so as to increase the adoption probability. On the contrary, for high-valuation users (those willing to pay the original price), it is wiser to recommend it before the price drops. This illustrates that sellers may benefit from taking full advantage of known price information in near future. Thus, *revenue-driven recommendations should be not only price-aware, but also sensitive to timing*. As such, we focus on a dynamic setting where a strategic recommendation plan is rolled out over a time horizon for which pricing information is available

\*An abridged version of this work is published in the Proceedings of the VLDB Endowment (PVLDB), volume 7, issue 14.

(either deterministically or probabilistically). We note that by myopically repeating items, the static approaches can be used to roll out recommendations over a horizon; we will empirically compare the performance of such strategies with our solutions in §6.

The dynamic setting naturally leads to two more important aspects related to revenue: competition and repeated recommendations. Products in the same class (kind) provide similar functionalities to customers, and thus an individual is unlikely to purchase multiple of them at once. This signals competition and its implications for the actual revenue yielded by repeated recommendations cannot be ignored. For instance, iPhone 5S, Nokia 820, and Blackberry Z10 are all smartphones. While an individual may find them all appealing, she is likely to purchase at most one even if all three phones are suggested to her (potentially more than once) during a short time period. In addition, as observed recently [8], while moderate repeated recommendation may boost the ultimate chance of adoption, overly repeating could lead to boredom, or *saturation*. All these factors, not considered in previous work [3, 6, 7], call for careful modeling and a more sophisticated algorithmic framework.

Driven by the above, we pose the problem of designing recommendation strategies to maximize the seller’s expected revenue as a novel discrete optimization problem, named **REVMAX** (for *revenue maximization*). Our framework allows any type of RS to be used, be it content-based, memory-based CF, or model-based CF. It takes prices as exogenous input to the RS, and nicely models adoption uncertainty by integrating the aforementioned factors: price, competition, and saturation (see §3 for a detailed justification and supporting literature). We show **REVMAX** is NP-hard and develop an approximation algorithm (for a relaxed version) and fast greedy heuristics. To the best of our knowledge, no existing work on RS has a revenue model that incorporates the various factors captured by our revenue model. Furthermore, the discrete optimization problem proposed and the algorithmic solution framework designed in this work are unique and complement previous work nicely.

To summarize, we make the following contributions.

- We propose a dynamic revenue model that accounts for time, price, saturation effects, and competition. Under this model, we propose **REVMAX**, the problem of finding a recommendation strategy that maximizes the seller’s expected total revenue over a short time horizon. To the best of our knowledge, this framework is novel and unique.
- We show that **REVMAX** is NP-hard (§3) and its objective function is non-monotone and submodular (§4). By establishing a connection to matroids, a relaxed version of **REVMAX** can be modeled as submodular function maximization subject to matroid constraints, enabling a local search algorithm with an approximation guarantee of  $1/(4 + \epsilon)$ , for any  $\epsilon > 0$  (§4).
- Since the local search approximation algorithm is prohibitively expensive and not practical, we design several clever and scalable greedy heuristics for **REVMAX** (§5).
- We perform a comprehensive set of experiments on two real datasets (Amazon and Epinions) that we crawled, to show the effectiveness and efficiency of our algorithms. In particular, they significantly outperform natural baselines. We use the data to learn the various parameters used in the experiments (§6). We also show that our best algorithm, Global Greedy, scales to a (synthetically generated) dataset that is 2.5 times as large as Netflix, the largest publicly available ratings dataset.
- We also discuss promising extensions to our framework in §7. In particular, we show that when exact price information is not available but only a price distribution is known, Taylor approximation can be employed for maximizing (approximate) expected total revenue.

## 2. BACKGROUND AND RELATED WORK

The majority of the research in RS has focused on algorithms for accurately predicting user preferences and their utility for items in the form of ratings, from a set of observed ratings. As mentioned in §1, RS algorithms can be classified into content-based or collaborative filtering (CF) methods, with CF being further classified into memory-based or model-based [1]. The state-of-the-art is collaborative filtering using Matrix Factorization (MF), which combines good accuracy and scalability [18]. In MF, users and items are characterized by latent feature vectors inferred from ratings. More specifically, we learn for each user  $u$  a vector  $\mathbf{p}_u \in \mathbb{R}^f$  and for each item  $i$  a vector  $\mathbf{q}_i \in \mathbb{R}^f$ , such that  $r_{ui} \approx \mathbf{p}_u^T \mathbf{q}_i$  for all observed ratings  $r_{ui}$  ( $f$  is the number of latent features). Training the model requires an appropriate loss function to measure the training error, so that using methods like stochastic gradient descent or alternating least squares, the latent feature vectors can be learned quickly in a few iterations [18]. The most common loss functions is RMSE [18]. For our purposes, existing MF methods can be used to compute predicted ratings accurately and efficiently, although our framework does permit the use of other methods. Recently, Koren [17] considers the drift in user interests and item popularity over time and proposes a temporal MF model to further improve accuracy. This model is for rating prediction and uses data spanning a *long* period, since drifts usually occur gradually, while our framework addresses dynamic recommendations over a *short* period and has a completely different objective (revenue).

Other related work of the customer-centric view includes [31, 32]. Zhao et al. [32] compute the time interval between successive product purchases and integrate it into a utility model for recommendations, with the idea of capturing the timing of recommendations well and increasing temporal diversity. Wang et al. [31] apply survival analysis to estimate from data the probability that a user will purchase a given product at a given time. It offers a way of modeling and computing adoption probability and is in this sense orthogonal to **REVMAX**. We depart from these works by focusing on strategic recommendation plans for short-term for which price information is known and repeated purchase is rare, and formulating **REVMAX** as a discrete optimization problem.

There has been some work on revenue-driven recommendations [3, 6, 7]. However, most of it deals with a *static* setting and is limited in several senses. Chen et al. [6] model adoption probabilities using purchase frequency and user similarity, and recommend each user  $k$ -highest ranked items in terms of expected revenue (probability  $\times$  price). Important ingredients such as capacity constraints, competition, price fluctuations and saturation are all ignored. Consequently, their optimization can simply be solved independently for each user, without the combinatorial explosion inherent in our problem, which is far more challenging. It is also tied to memory-based RS (user similarity), whereas our approach is more generic and works for any type of RS. Das et al. [7] propose a simple profit model which predicts the adoption probability using the similarity between the user’s true ratings and the system prediction. It captures the trust between users and the RS, but does not take into account the fact that when the price or relevance of suggested items changes, user’s adoption probability ought to change too. Azaria et al. [3] model the probability of a movie being viewed based on its price and rank (from a black-box RS). Their setting is also static and focuses on movies only, which is not sufficient for most e-commerce businesses that sell a variety of products.

Static revenue-driven recommendation, on an abstract level, is related to generalized bipartite matching [10, 11]. The papers [21, 23] propose distributed solution frameworks to scale this up to massive datasets. They are substantially different from and or-

thogonal to ours as the focus is on distributed computation, still within a static setting. We defer detailed discussion to §3.

### 3. REVENUE MAXIMIZATION

The motivation for a dynamic view of revenue-based recommendations is twofold. First, prices change frequently, sometimes even on a daily basis, or several times a day in online marketplaces like Amazon [2, 4]. A static recommendation strategy is forced to be myopic and cannot take advantage of price fluctuations to choose the right *times* at which an item should be recommended. This shortcoming is glaring, since for product adoptions to occur, a user’s *valuation* of an item, i.e., the maximum amount of money she is willing to pay, must exceed the price. This is a well-known notion in economics literature [15, 24, 28]. And obviously, the optimal time in terms of extracting revenue is not necessarily when the price is at its peak or bottom! Therefore, *only by incorporating time, we can design revenue-maximizing recommendations in a systematic, holistic, and principled way.*

Second, we can recommend items strategically over a time horizon, possibly making some repetitions, to improve the chances that some items are eventually bought. However, this is delicate. Indiscriminate repeat recommendations can lead to *saturation* effects (boredom), turning off the user from the item or even costing her loyalty to the RS. Thus, we need to choose the *timing* of recommendations as well as their *frequency* and *placement* so as to maximize the expected revenue in the presence of saturation effects. *These opportunities are simply not afforded when recommendations are made in a static, one-shot basis.*

#### 3.1 Problem Setting and Definition

We adopt a discrete time model to describe the dynamics of revenue in an e-commerce business. Let  $U$  and  $I$  be the set of users and items respectively, and define  $[T] := \{1, 2, \dots, T\}$  to be a short time horizon over which a business (or seller) plans a recommendation strategy. The seller may choose the granularity of time at her discretion. E.g., the granularity may be a day and the horizon may correspond to a week.

Naturally, items can be grouped into classes based on their features and functionalities, e.g., “Apple iPad Mini” and “Google Nexus 7” belong to class `tablet`, while “Samsung Galaxy S4” and “Google Nexus 5” belong to `smartphone`. Items in the same class offer a variety of options to customers and thus compete with each other, since from an economic point of view, it is unlikely for a person to adopt multiple products with similar functionality over a short time horizon. Thus in our model, we assume items within a class are competitive in the sense that their adoption by any user within  $[T]$  is *mutually exclusive*.

In general, revenue to be expected from a recommendation depends mainly on two factors: the price of the item, and the probability that the user likes and then adopts the item at that price. The latter, as mentioned in §1, is in turn dependent on many factors, including but not restricted to, price. The former, price, is conceptually easier to model, and we suggest two alternatives. Both of the following essentially model prices to be *exogenous* to the RS, meaning, they are externally determined and provided to the RS. First, we can use an *exact price* model which assumes that for each item  $i$  and time step  $t$ , the exact value  $p(i, t)$  is known. Exact pricing is studied extensively in microeconomics theory. There is a well established theory for estimating future demand and supply, which are further used to derive future price in market equilibrium [30]. In practice, we also observe that retailers (both online and offline) do plan their pricing ahead of time (e.g., sales on Black Friday in the US and Boxing Day in Canada). Alternatively, in case prices

are not completely known – which could be possible if the time of interest is too far into the future, or there isn’t sufficient data to determine a precise value – they can be modeled as random variables following a certain probability distribution. In other words, there is a price prediction model that produces a probability distribution associated with price. As we shall see shortly, the challenge of revenue maximization in RS remains in both exact price and random price models. In the bulk of this paper, we focus on the exact price model, but in §7, give insights on how the random price model can be incorporated into our framework.

It is well known that there is uncertainty in product adoption by users. Consider any user-item-time triple  $(u, i, t) \in U \times I \times [T]$ . We associate it with a (*primitive*) *adoption probability*  $q(u, i, t) \in [0, 1]$ , the probability that  $u$  would purchase  $i$  at time  $t$ . Based on economics theory, buyers are rational and utility-maximizing [24, 28] and would prefer lower price for higher utility (of consuming a product). Rational agents are typically assumed to possess a private, internal *valuation* for a good, which is the maximum amount of money the buyer is willing to pay; furthermore, users are assumed to be price-takers who respond myopically to the prices offered to them, solely based on their privately-held valuations and the price offered. Both the internal private valuation assumption and the price taker assumption are standard in economics literature [24, 28]. Thus, though a lower price leads to a higher probability of adoption, it might not be optimal to recommend an item at its lowest price, since some buyers actually could be willing to pay more. In general, a seller may estimate the adoption probabilities using its own data and model, and the exact method used for estimating adoption probabilities is orthogonal to our framework. In §6, we give one method for estimating adoption probabilities from the real datasets Amazon and Epinions. We note that the above notion is “primitive” in the sense that it ignores the effect of competition and saturation, which will be captured in Definition 1.

Intuitively, the expected revenue yielded by recommending item  $i$  to user  $u$  at time  $t$  is  $p(i, t) \times q(u, i, t)$ , if this recommendation is considered in isolation. However, competition and repeated suggestions make the expected revenue computation more delicate. We first formally define a *recommendation strategy* as a set of user-item-time triples  $S \subseteq U \times I \times [T]$ , where  $(u, i, t) \in S$  means  $i$  is recommended to  $u$  at time  $t$ . Let us next consider the effect of competition and repeated recommendations. Suppose  $S$  suggests multiple items (possibly repeated) within a class to user  $u$ , then the adoption decisions that a user makes at different times are *interdependent* in the following manner: The event that a user  $u$  adopts an item  $i$  at time  $t$  is conditioned on  $u$  not adopting any item from  $i$ ’s class before, and once  $u$  adopts  $i$ , she will not adopt anything from that class again in the horizon (which, it may be recalled, is short in our model). This semantics intuitively captures competition effects.

As argued earlier, repeated suggestions may lead to a boost in adoption probability, but repeating too frequently may backfire, as people have a tendency to develop boredom, which can potentially cause devaluation in user preference [8, 16]. We call this *saturation effect*. This is undesirable in terms of revenue and thus the model should be saturation-aware. We discount the adoption probability by accounting for saturation, similar in spirit to [8]. The intuition is that the more often and more recently a user has been recommended an item or an item from the same class, the fresher her memory and consequently the more significant the devaluation. Let  $\mathcal{C}(i)$  denote the class to which  $i$  belongs. The memory of user  $u$  on item  $i$  at any time  $t > 1$  w.r.t. a recommendation strategy  $S$  is

$$M_S(u, i, t) := \sum_{j \in \mathcal{C}(i)} \sum_{\tau=1}^{t-1} \frac{X_S(u, j, \tau)}{t - \tau}, \quad (1)$$



where  $X_S(u, j, \tau)$  is an indicator variable taking on 1 if  $(u, j, \tau) \in S$  and 0 otherwise. Also  $X_S(u, i, 1) := 0$  for all  $u, i$ , and  $S$ . The discounting on adoption probability should reflect the fact that devaluation is monotone in memory. More specifically, given user  $u$ , item  $i$ , and time step  $t$ , we define the *saturation-aware adoption probability* under a strategy  $S$  as  $q(u, i, t) \times \beta_i^{M_S(u, i, t)}$ , where  $\beta_i \in [0, 1]$  is the saturation factor of item  $i$ . This effectively penalizes recommending the same class of items in (nearly) consecutive time steps. Smaller  $\beta_i$  means greater saturation effect.  $\beta_i = 1$  corresponds to no saturation and  $\beta_i = 0$  corresponds to full saturation: any repetition immediately leads to zero probability. In principle,  $\beta_i$ 's can be learned from historical recommendation logs (cf. [8]).

Combining competition and saturation effects, we are now ready to define the *strategy-dependent dynamic adoption probability*. Basically, the dynamic adoption probability for later times depends on earlier times at which recommendations are made. Intuitively,  $q_S(u, i, t)$  is the probability that  $u$  adopts  $i$  at time  $t$  and does not adopt any item from its class earlier, under the influence of  $S$ .

**DEFINITION 1 (DYNAMIC ADOPTION PROBABILITY).** *For any user  $u$ , item  $i$ , and time step  $t$ , given a strategy  $S$ , the dynamic adoption probability governed by  $S$  is defined as follows:*

$$q_S(u, i, t) = q(u, i, t) \times \beta_i^{M_S(u, i, t)} \times \prod_{\substack{(u, j, \tau) \in S: \\ j \neq i, C(j) = C(i)}} (1 - q(u, j, \tau)) \prod_{\substack{(u, j, \tau) \in S: \\ \tau < t, C(j) = C(i)}} (1 - q(u, j, \tau)). \quad (2)$$

Also, define  $q_S(u, i, t) = 0$  whenever  $(u, i, t) \notin S$ .

**EXAMPLE 1 (DYNAMIC ADOPTION PROBABILITY).** *Suppose the strategy  $S$  consists of  $\{(u, i, 1), (u, j, 2), (u, i, 3)\}$ , where  $C(i) = C(j)$ , and for all three triples, the (primitive) adoption probability is  $a$ . Then,  $q_S(u, i, 1) = a$ ,  $q_S(u, j, 2) = (1 - a) \cdot a \cdot \beta_i^{\frac{1}{2}}$ , and  $q_S(u, i, 3) = (1 - a)^2 \cdot a \cdot \beta_i^{\frac{1}{2} + \frac{1}{2}}$ .*

We next define the *expected revenue* of a recommendation strategy, as the expected amount of money the business (seller) can earn from recommendation-led adoptions by following the strategy.

**DEFINITION 2 (REVENUE FUNCTION).** *Given a set  $U$  of users, a set  $I$  of items, and time horizon  $[T]$ , the expected revenue of any strategy  $S \subseteq U \times I \times [T]$  is*

$$Rev(S) = \sum_{(u, i, t) \in S} p(i, t) \times q_S(u, i, t), \quad (3)$$

where  $q_S(u, i, t)$  is defined in Definition 1.

The *revenue maximization problem* asks to find a strategy  $S$  with maximum expected revenue, subject to the following two natural constraints: First, the *display constraint*, following standard practice, requires that no more than  $k$  items be recommended to a user at a time. Second, as items all have limited quantities in inventory, we impose a *capacity constraint*: no item  $i$  may be recommended to more than  $q_i$  distinct users at any time, where  $q_i$  is a number determined based on current inventory level and demand forecasting [25]. The intention is to avoid potential customer dissatisfaction caused by out-of-stock items. In general,  $q_i$  can be somewhat higher than the actual inventory level, due to uncertainty in product adoption. We call a strategy  $S$  *valid* if  $S$  satisfies the following constraints: (i) for all  $u \in U$  and all  $t \in [T]$ ,  $|\{i \mid (u, i, t) \in S\}| \leq k$ ; (ii) for all  $i \in I$ ,  $|\{u \mid \exists t \in [T] : (u, i, t) \in S\}| \leq q_i$ .

**PROBLEM 1 (REVENUE MAXIMIZATION (REVMAX)).** *Given a set  $U$  of users,  $I$  of items, time horizon  $[T]$ , display*

*limit  $k$ ; for each  $i \in I$ , capacity  $q_i$ , price  $p(i, t) \forall t \in [T]$ , and saturation factor  $\beta_i$ ; adoption probabilities  $q(u, i, t)$  for all  $(u, i, t) \in U \times I \times [T]$ , find the optimal valid recommendation strategy, i.e.,  $S^* \in \arg \max_{S \text{ is valid}} Rev(S)$ .*

## 3.2 Hardness of Revenue Maximization

As we will show shortly, REVMAX is in general NP-hard. However, when  $T = 1$ , it is PTIME solvable since it can be cast as a special case of the *maximum-weighted degree-constrained subgraph* problem (Max-DCS) [10, 11]. In Max-DCS, we are given a graph  $G = (V, E)$ , a weight  $w_e$  for each edge  $e \in E$ , and a number  $d_v$  for each node  $v \in V$ . The task is to find a subgraph of  $G$  that has maximum weight and for each  $v \in V$ , it contains no more than  $d_v$  edges incident to  $v$ . The best known combinatorial algorithm for Max-DCS takes time  $O(D(|E| + |V| \log |V|))$  [10, 11].

Given an instance  $\mathcal{I}$  of REVMAX with  $T = 1$ , we create an instance  $\mathcal{J}$  of Max-DCS, by building a bipartite graph  $G' = (U \cup I, E')$ , where  $E' \subseteq U \times I$ . We create one node per user, and per item. For each user-node  $u$ , set  $d_u = k$ , and for each item-node  $i$ , set  $d_i = q_i$ . There is an edge  $(u, i) \in E'$  if  $q(u, i, 1) > 0$  and set the weight  $w_{(u, i)} = p(i, 1) \times q(u, i, 1)$ . It is easy to see the optimal subgraph in  $\mathcal{J}$  corresponds to the optimal strategy in  $\mathcal{I}$ . Thus, this special case of REVMAX can be solved efficiently using known algorithms for Max-DCS.

We now show that the general REVMAX is NP-hard. The reduction is from a restricted version of the Timetable-Design problem (RTD), which is NP-complete [9, 13], to the decision-version of REVMAX (D-REVMAX). An instance of the RTD problem consists of a set  $C$  of craftsmen, a set  $B$  of jobs, and a set of  $H$  of hours, where  $|H| = 3$ . Each craftsman  $c \in C$  is available for a subset  $A(c) \subseteq H$  of hours, thus  $|A(c)|$  is the number of hours that  $c$  is available. There is a function  $R : C \times B \rightarrow \{0, 1\}$ , such that  $R(c, b)$  specifies the number of hours craftsman  $c$  is required to work on job  $b$ . A craftsman is a  $\tau$ -craftsman if  $|A(c)| = \tau$ . Furthermore, he is *tight*, if he is a  $\tau$ -craftsman for some  $\tau$  and is required to work on  $\tau$  jobs, i.e.,  $\sum_{b \in B} R(c, b) = |A(c)| = \tau$ . In RTD, every craftsman is either a 2-craftsman or a 3-craftsman and is tight. A timetable is a function  $f : C \times B \times H \rightarrow \{0, 1\}$ . It is *feasible* if all of the following hold:

1.  $f(c, b, h) = 1$  implies  $h \in A(c)$ ;
2. for each  $h \in H$  and  $c \in C$  there is at most one  $b \in B$  such that  $f(c, b, h) = 1$ ;
3. for each  $h \in H$  and  $b \in B$  there is at most one  $c \in C$  such that  $f(c, b, h) = 1$ ; and
4. for each pair  $(c, b) \in C \times B$  there are exactly  $R(c, b)$  values of  $h$  for which  $f(c, b, h) = 1$ .

The task in RTD is to determine if a feasible timetable exists.

**THEOREM 1.** *Problem 1 (REVMAX) is NP-hard.*

**PROOF.** Given an instance  $\mathcal{I}$  of RTD, we create an instance  $\mathcal{J}$  of D-REVMAX as follows. Craftsmen and hours in  $\mathcal{I}$  correspond to users and time steps in  $\mathcal{J}$ . For each job  $b$ , create three items of class  $b$ :  $i_1^b, i_2^b, i_3^b$ . The subscripts correspond to the three hours in  $H$ . For every item the capacity constraint is 1, i.e.,  $q(i_\tau^b) = 1, \forall b \in B, \tau \in H$ . The display constraint  $k = 1$ . Set price  $p(i_\tau^b, t) = 1$  if  $t = \tau$ , and 0 otherwise. For user  $c$ , item  $i_\tau^b$ , and any time step  $t \in H$ , set  $q(c, i_\tau^b, t) = R(c, b)$ . For every user  $c \in C$ , we create a unique expensive item  $e_c$  and set  $p(e_c, t) = E$  for all time  $t$ , where  $E$  is an arbitrary number such that  $E > N := \sum_{c \in C, b \in B} R(c, b)$ .

For any time at which  $c$  is unavailable, i.e., if  $t \in H \setminus A(c)$ , set  $q(c, e_c, t) = 1$  and for all times at which  $c$  is available, i.e., for  $t \in A(c)$ ,  $q(c, e_c, t) = 0$ . Finally, let  $\Upsilon = \sum_{c \in C} |H \setminus A(c)|$ , i.e., the total number of unavailable hours of all craftsmen.

CLAIM:  $\mathcal{J}$  admits a recommendation strategy of expected revenue  $\geq N + \Upsilon E$  if and only if  $\mathcal{I}$  admits a feasible timetable.

( $\Leftarrow$ ): Suppose  $f$  is a timetable for  $\mathcal{I}$ . Define the recommendation strategy as  $S = \{(c, i_t^b, t) \mid f(c, b, t) = 1\} \cup \{(c, e_c, t) \mid t \in H \setminus A(c)\}$ . It is straightforward to verify that every recommendation will lead to an adoption, including the expensive items. Notice that by feasibility of  $f$ , no craftsman is assigned to more than one job at a time and no job is assigned to more than one craftsman at any time, so the display and capacity constraints are satisfied by  $S$ . It follows that the expected revenue is exactly  $N + \Upsilon E$ .

( $\Rightarrow$ ): Let  $S$  be a valid recommendation strategy that leads to an expected revenue  $\geq N + \Upsilon E$ . Unless every recommendation in  $S$  leads to adoption, the expected revenue cannot reach  $N + \Upsilon E$ . This implies that  $S$  cannot recommend more than one item from a certain class to any user, i.e.,  $S$  cannot contain both  $(c, i_{t_1}^b, t_1)$  and  $(c, i_{t_2}^b, t_2)$  for any user  $c$ , class  $b$  and times  $t_1 \neq t_2$ , for if it did, one of the recommendations (the one that occurred later) would be rejected by user  $c$  for sure, so the opportunity to make a profit at that future time is lost. Define  $f(c, b, t) = 1$  whenever  $(c, i_t^b, t) \in S$ , and  $f(c, b, t) = 0$  otherwise. We will show that  $f$  is a feasible timetable. By construction, expensive items do not correspond to jobs and must have been recommended to each user  $c$  when  $c$  is unavailable according to  $A(c)$ . Any optimal strategy will contain these expensive item recommendations which account for a profit of  $\Upsilon E$ . Thus, the remaining recommendations in  $S$  must net an expected revenue of at least  $N$ . Since each of them recommends an inexpensive item corresponding to a job, it can lead to an expected revenue of at most 1. We shall next show that  $f$  is feasible. (1) Let  $f(c, b, t) = 1$ . Then we have  $(c, i_t^b, t) \in S$  by construction. Since every recommendation must make a non-zero revenue in an optimal strategy, we must have  $t \in A(c)$  by construction. (2) Suppose  $f(c, b, t) = f(c', b, t) = 1$ , for some distinct users  $c, c'$ . This implies  $(c, i_t^b, t), (c', i_t^b, t) \in S$ . But this is impossible, since  $q(i_t^b) = 1$ , which would make  $S$  invalid. (3) Suppose  $f(c, b, t) = f(c, b', t) = 1$ , for some distinct jobs  $b, b'$ . This is impossible as it would make  $S$  violate the display constraint  $k = 1$ , as it would recommend both  $i_t^b$  and  $i_t^{b'}$  to user  $c$  at time  $t$ . (4) Finally, we shall show that for each  $(c, b) \in C \times B$ , there are exactly  $R(c, b)$  hours,  $t$ , for which  $f(c, b, t) = 1$ . We will prove this by contradiction. The key observation is that every recommendation in  $S$  must make a non-zero revenue. Specifically,  $N$  of them should make a revenue of 1 each and  $\Upsilon$  of them should make a revenue of  $E$  each. It is easy to see that if this is not the case, then  $\pi(S) < N + \Upsilon E$ .

The following cases arise.

Case 1:  $R(c, b) = 1$ . Suppose  $f$  does not assign  $c$  to job  $b$  even once. In this case, either  $S$  has fewer than  $N$  recommendations making a revenue of 1 each or has a recommendation making no profit. In both cases,  $\pi(S) < N + \Upsilon E$ . Suppose  $f$  assigns  $b$  to  $c$  more than once, say  $f(c, b, t) = f(c, b, t') = 1$ , and assume w.l.o.g. that  $t < t'$ . Then  $S$  must contain the recommendations, say  $(c, i_t^b, t)$  and  $(c, i_{t'}^b, t')$ . By construction, however,  $q_S(c, i_{t'}^b, t') = 0$  so this is a wasted recommendation. Thus,  $\pi(S) < N + \Upsilon E$ .

Case 2:  $R(c, b) = 0$ . Suppose  $f(c, b, t) = 1$  for some  $t \in H$ . This implies  $(c, i_t^b, t) \in S$ , but  $q_S(c, i_t^b, t) = 0$  by construction, which again makes it a wasted recommendation, so  $\pi(S) < N + \Upsilon E$ .

This shows that the timetable function  $f$  is indeed feasible, which completes the proof.  $\square$

We stress that the proof does not involve saturation, which means

even when the saturation parameter  $\beta_i = 1$  for all  $i \in I$  (no saturation at all!), REVMAX remains NP-hard.

## 4. THEORETICAL ANALYSIS & APPROXIMATION ALGORITHM

Given the hardness of REVMAX, a natural question is whether we can design an approximation algorithm. In this section, we present an approximation algorithm by establishing connections to the problem of *maximizing non-negative submodular functions subject to a matroid constraint*. To this end, we first show that the revenue function  $Rev(\cdot)$  is submodular (albeit non-monotone). Next, we transform the display constraint of REVMAX into a *partition matroid* constraint. Finally, we “push” a “smoothened” version of the capacity constraint into the objective function. This results in a version of the REVMAX problem that has a relaxed capacity constraint. Solving it reduces to maximizing a non-negative non-monotone submodular function under a matroid constraint. This can be approximated to within a factor of  $1/(4 + \epsilon)$ , for any  $\epsilon > 0$ , achieved by applying a local search algorithm due to Lee et al. [19].

### 4.1 Submodularity of Revenue Function

We show that the revenue function  $\pi(\cdot)$  is submodular. Let  $X$  be any finite ground set. A non-negative set function  $f : 2^X \rightarrow \mathbb{R}_{\geq 0}$  is *monotone* if  $f(S) \leq f(S')$  whenever  $S \subseteq S' \subseteq X$ . Also,  $f$  is *submodular* if for any two sets  $S \subseteq S' \subseteq X$  and any  $w \in X \setminus S'$ ,  $f(S \cup \{w\}) - f(S) \geq f(S' \cup \{w\}) - f(S')$ . Submodularity captures the principle of *diminishing marginal returns* in economics.  $Rev(\cdot)$  is nonnegative by definition. Next, we describe how to compute the marginal revenue of a triple  $z = (u, i, t)$  w.r.t. a strategy  $S$ , denoted  $Rev_S(z) := Rev(S \cup \{z\}) - Rev(S)$ . This is useful both for showing submodularity and illustrating algorithms in §5.

The marginal revenue  $z$  brings to  $S$  consists of two parts: a *gain* of  $q_{S \cup \{z\}}(z) \cdot p(i, t)$  from  $z$  itself, and a *loss* as adding  $z$  to  $S$  drops the dynamic adoption probability of triples  $(u, j, t') \in S$  with  $j \in \mathcal{C}(i)$  and  $t' > t$ . More specifically,  $q_{S \cup \{z\}}(u, j, t') \leq q_S(u, j, t')$  due to the increased memory caused by  $z$  and the interdependency rule. The exact formula for computing  $Rev_S(z)$  is as follows.

DEFINITION 3 (MARGINAL REVENUE). *Given any valid strategy set  $S$  and a triple  $z = (u, i, t) \notin S$ , the marginal revenue of  $z$  with respect to  $S$  is*

$$Rev_S(z) = p(i, t) \times q_{S \cup \{z\}}(z) + p(i, t') \times \sum_{\substack{(u, j, t') \in S \\ j \in \mathcal{C}(i), t' > t}} (q_{S \cup \{z\}}(u, j, t') - q_S(u, j, t')). \quad (4)$$

It can be shown that the revenue function satisfies submodularity.

THEOREM 2. *The revenue function  $Rev(\cdot)$  is non-monotone and submodular w.r.t. sets of user-item-time triples.*

First, we give a useful lemma that will be used later for proving Theorem 2.

LEMMA 1. *Given any triple  $(u, i, t) \in S$ , its dynamic adoption probability  $q_S(u, i, t)$  is non-increasing in the strategy set  $S$ . That is, given two strategy sets  $S, S'$  such that  $(u, i, t) \in S \subseteq S'$ , we have  $q_S(u, i, t) \geq q_{S'}(u, i, t)$ .*

PROOF. By definition, all triples with  $t' > t$  do not affect dynamic adoption probability at  $t$ . Since every  $(u, j, *)$ -triple with  $t' < t$  and  $j \in \mathcal{C}(i)$  in  $S$  is also in  $S'$ , by Equation (2), the lemma follows naturally.  $\square$

PROOF OF THEOREM 2. For non-monotonicity, consider an instance which has  $U = \{u\}$ ,  $I = \{i\}$ ,  $T = 2$ ,  $k = 1$ ,  $q_i = 2$ ,  $q(u, i, 1) = 0.5$ ,  $q(u, i, 2) = 0.6$ ,  $p(i, 1) = 1$ ,  $p(i, 2) = 0.95$ , and  $\beta_i = 0.1$ . Consider two sets  $S = \{(u, i, 2)\}$  and  $S' = \{(u, i, 1), (u, i, 2)\}$ . Clearly  $Rev(S') = 0.5285 < Rev(S) = 0.57$ , implying that  $Rev(\cdot)$  is non-monotone.

For submodularity, consider two strategies  $S \subseteq S'$ , and a triple  $z = (u, i, t) \notin S'$ . We say that  $z$  *succeeds* triple  $z' = (u', i', t')$  if  $t > t'$ , or *precedes*  $z'$  if  $t < t'$ . The following cases arise.

**Case 1:**  $z$  succeeds all  $(u, j)$ -triples in  $S'$  such that  $C(j) = C(i)$ . Since  $S \subseteq S'$ ,  $z$  succeeds all such triples in  $S$ , too, and thus adding  $z$  to  $S$  or  $S'$  will not cause any loss in revenue. By Lemma 1,  $q_{S' \cup \{z\}}(z) \leq q_{S \cup \{z\}}(z)$ , and thus  $Rev_S(z) \geq Rev_{S'}(z)$ .

**Case 2:**  $z$  precedes all  $(u, j)$ -triples in  $S'$  such that  $C(j) = C(i)$ . Since  $S \subseteq S'$ ,  $z$  also precedes all such triples in  $S$ , in which case  $z$  brings the same amount of revenue gain to both sets since  $q_{S \cup \{z\}}(z) = q_{S' \cup \{z\}}(z) = q(z)$ . However, the number of triples  $z$  precedes in  $S'$  is no less than that in  $S$ , so is the revenue loss  $z$  causes. Thus,  $Rev_S(z) \geq Rev_{S'}(z)$ .

**Case 3:**  $z$  precedes a few  $(u, j)$ -triples in  $S'$  where  $C(j) = C(i)$  and succeeds the rest of such triples in  $S'$ . The argument for this case combines the reasoning for the two cases above. First, for revenue gain, by Lemma 1,  $q_{S' \cup \{z\}}(z) \leq q_{S \cup \{z\}}(z)$ . Second, for revenue loss, the number of triples  $z$  precedes in  $S'$  is no less than that in  $S$ , so is the revenue loss. Hence  $Rev_S(z) \geq Rev_{S'}(z)$  holds.

This completes the proof.  $\square$

Theorem 2 not only helps us in the approximation analysis below, but also lays the foundation for the lazy forward technique used in our greedy algorithm in Section 5.

## 4.2 Matroid and Approximation Guarantees

Matroids are a family of abstract structures widely studied in combinatorics. A *matroid*  $\mathcal{M}$  is a pair  $(X, \mathcal{I})$  where  $X$  is a finite ground set and  $\mathcal{I} \subseteq 2^X$  is a collection of subsets of  $X$  called *independent sets*, whose membership satisfies: (1).  $\emptyset \in \mathcal{I}$ ; (2). Downward closure:  $\forall T \in \mathcal{I}$ ,  $S \subset T$  implies  $S \in \mathcal{I}$ ; (3). Augmentation:  $\forall S, S' \in \mathcal{I}$ , if  $|S| < |S'|$ , then  $\exists w \in S' \setminus S$  s.t.  $S \cup \{w\} \in \mathcal{I}$ .

The problem of maximizing  $f : 2^X \rightarrow \mathbb{R}_+$  subject to a matroid constraint  $\mathcal{M} = (X, \mathcal{I})$  is to find  $S^* \in \arg \max_{S \in \mathcal{I}} f(S)$ , which is in general NP-hard for nonnegative submodular functions [19].

We deal with the display constraint first. A *partition matroid*  $(X, \mathcal{I})$  is defined by  $m$  disjoint subsets of  $X$  with  $\cup_{j=1}^m X_j = X$ , with each  $X_j$  having a maximum cardinality constraint  $b_j$ .  $\mathcal{I}$  contains  $S \subseteq X$  iff  $|S \cap X_j| \leq b_j$  for all  $j$ .

LEMMA 2. *The display constraint in REVMAX is equivalent to a partition matroid constraint.*

PROOF. Let the ground set  $X = U \times I \times [T]$ . We project  $X$  onto all user-time pairs  $(u^*, t^*) \in U \times [T]$  to obtain a collection of subsets  $X(u^*, t^*) := \{(u, i, t) | u = u^*, i \in I, t = t^*\}$ . Clearly, the sets  $X(u, t)$ ,  $u \in U, t \in [T]$ , are pairwise disjoint and  $\cup_{u,t} X(u, t) = X$ . Next, set  $b(u, t) = k$  for all  $(u, t)$ . This gives a partition matroid  $\mathcal{M} = (X, \mathcal{I})$  and any  $S \subseteq X$  satisfies the display constraint iff  $S \in \mathcal{I}$ .  $\square$

Unlike the display constraint, the capacity constraint does not correspond to a matroid constraint. The observation is that while downward closure and inclusion of empty set are satisfied, augmentation is not satisfied by the capacity constraint, as we next show with an example.

EXAMPLE 2 (CAPACITY CONSTRAINT). *Consider two strategies  $S' = \{(u_1, i_2, t_1), (u_1, i_2, t_2), (u_2, i_1, t_1), (u_2, i_1, t_2)\}$*

*and  $S = \{(u_1, i_1, t_1), (u_2, i_2, t_2)\}$ . Assume the display constraint is  $k = 1$  and the capacity constraint is  $q_{i_1} = q_{i_2} = 1$ . While  $|S'| > |S|$ , there is no triple in  $S' \setminus S$  which can be added to  $S$  without violating the capacity constraint.*

Thus, we resort to a different method for obtaining an approximation algorithm. In REVMAX, the capacity constraint is a hard constraint: no item  $i$  can be recommended to more than  $q_i$  distinct users. This can in principle result in fewer than  $q_i$  adoptions, because of the inherent uncertainty involved. Consider making a calculated move in making a few more recommendations than the capacity would allow, should all recommended users wish to adopt. Since they are not all guaranteed to adopt, it is possible that such a strategy may result in a greater expected profit than a strategy that REVMAX would permit. We next formalize this idea.

Let  $S$  be any strategy and  $(u, i, t) \in S$  be a triple. If  $i$  is not recommended to more than  $q_i$  distinct users up to time  $t$ ,  $q_S(u, i, t)$  should remain exactly the same as in (2). Suppose  $i$  has been recommended to  $\geq q_i$  users *besides*  $u$  up to time  $t$ . Two cases arise w.r.t. adoptions. If  $q_i$  of the previously recommended users adopt  $i$ , then we must conclude  $u$  cannot adopt it since it is not available any more! On the other hand, if fewer than  $q_i$  previously recommended users adopt  $i$ , then  $u$  may adopt  $i$  with probability  $q_S(u, i, t)$ . We refer to this relaxed version of REVMAX as  $R$ -REVMAX. Compared to REVMAX,  $R$ -REVMAX has no hard capacity constraint on number of recommendations of an item. This constraint is essentially “pushed” inside the *effective dynamic adoption probability* for  $R$ -REVMAX, defined as follows.

DEFINITION 4 (EFFECTIVE DYNAMIC ADOPTION PROB.).

*Let  $S$  be a strategy and suppose  $(u, i, t) \in S$  is a triple and let  $S_{i,t} = \{(v, i, \tau) \in S \mid v \neq u, \tau \leq t\}$  be the set of recommendations of  $i$  made to users other than  $u$  up to time  $t$ . Let  $\mathcal{B}_S(i, t) = \Pr[\text{at most } (q_i - 1) \text{ users in } S_{i,t} \text{ adopt } i]$ . Then the effective dynamic adoption probability of  $(u, i, t)$  given  $S$  is:*

$$\begin{aligned} \mathcal{E}_S(u, i, t) &:= q(u, i, t) \times \beta_i^{M_S(u, i, t)} \times \prod_{\substack{(u, j, t) \in S \\ j \neq i, C(j) = C(i)}} (1 - q(u, j, t)) \\ &\times \prod_{(u, j, \tau) \in S, \tau < t, C(j) = C(i)} (1 - q(u, j, \tau)) \times \mathcal{B}_S(i, t) \end{aligned} \quad (5)$$

The following example illustrates Definition 4.

EXAMPLE 3 (EXCEEDING CAPACITY). *Consider one item  $i$ , three users  $u, v, w$ , display constraint  $k = 1$ , capacity  $q_i = 1$ , and saturation parameter  $\beta_i = 0.5$ . Consider the strategy  $S = \{(u, i, 1), (v, i, 2), (w, i, 1), (w, i, 2)\}$ . Then the effective dynamic adoption probability of  $(w, i, 2)$  is  $\mathcal{E}(w, i, 2) = q(w, i, 2) \times (1 - q(w, i, 1)) \times (1 - q(u, i, 1)) \times (1 - q(v, i, 2)) \times 0.5^{1/1}$ .*

The only change to the definition of  $\mathcal{E}_S(u, i, t)$  in (5) compared with that of  $q_S(u, i, t)$  is the factor  $\mathcal{B}_S(i, t)$ . If  $|S_{i,t}| < q_i$ , this probability is 1. Computing it exactly in the general case can be hard but can be computed exactly in worst-case exponential time in  $q_i$ . We can use Monte-Carlo simulation for estimating this probability. The point is that given an oracle for estimating or computing probability, we can define  $R$ -REVMAX as follows.

An instance of  $R$ -REVMAX is identical to that of REVMAX without a hard capacity constraint. The revenue function  $Rev(S)$  is defined exactly as in (3), except that instead of  $q_S(u, i, t)$ ,  $\mathcal{E}_S(u, i, t)$  is used. A strategy is now called valid if it satisfies the display constraint, and the problem is to find a strategy  $S^*$  that maximizes  $Rev(S)$  amongst all valid strategies  $S$ .



Given an oracle for computing or estimating  $\mathcal{B}_S(i, t)$ , we can devise an approximation algorithm for  $R$ -REVMAX as follows. As shown in Lemma 2, the display constraint corresponds to a matroid constraint. The revenue function  $Rev(S)$  for  $R$ -REVMAX can be easily shown to be non-monotone and submodular. Thus, solving  $R$ -REVMAX corresponds to maximizing a non-negative non-monotone submodular function subject to a partition matroid constraint. This can be solved using local search to give a  $1/(4 + \epsilon)$ -approximation, for any  $\epsilon > 0$  [19]. However, unfortunately, the time complexity of this approximation algorithm, even assuming unit cost for each invocation of the oracle, is  $O(\frac{1}{\epsilon}|X|^4 \log |X|)$  where  $X = U \times I \times [T]$  in our case. This is prohibitive for our application and motivates the quest for good and efficient heuristics that perform well in practice, a topic addressed in the next section.

## 5. GREEDY ALGORITHMS

We propose three intelligent greedy algorithms for REVMAX: *Global Greedy* (G-Greedy), *Sequential Local Greedy* (SL-Greedy), and *Randomized Local Greedy* (RL-Greedy). They all start with an empty strategy set and incrementally grow it in a greedy manner. As we shall see shortly, the main difference is that G-Greedy operates on the entire ground set  $U \times I \times [T]$  and makes recommendations disregarding time order, while SL-Greedy and RL-Greedy finalize recommendations in a predetermined chronological order.

### 5.1 The Global Greedy Algorithm

**Overview.** We first give a natural hill-climbing style algorithm called Global Greedy (G-Greedy for short; pseudo-code in Algorithm 1). In a nutshell, the algorithm starts with  $S$  being  $\emptyset$ , and in each iteration, it adds to  $S$  the triple that provides the largest positive marginal revenue w.r.t.  $S$  without violating the display or capacity constraint. More formally, let  $\mathcal{V}(S) \subset U \times I \times [T]$  be the set of triples which, when added to  $S$ , would *not* violate the display or capacity constraints. Thus, in every iteration, G-Greedy selects the triple satisfying:

$$z^* \in \arg \max \{ Rev_S(z) > 0 \mid z \in \mathcal{V}(S) \setminus S \}. \quad (6)$$

Recall that  $Rev_S(z)$  represents the marginal revenue of  $z$  w.r.t.  $S$ , defined as  $Rev(S \cup \{z\}) - Rev(S)$  (cf. Equation (4)). Also, we use priority queues in the implementation to support efficient operations in the greedy selection process.

To enhance efficiency, we also employ two implementation-level optimizations. First, we propose the idea of *two-level heaps* data structure to reduce the overhead of heap operations in the greedy selection process. Second, the *lazy forward* scheme [20, 22] is used for computing and ranking triples w.r.t. their marginal revenue.

The algorithm also maintains several auxiliary variables to facilitate constraint enforcement. First, counter variables are used to keep track of the number of items recommended to each user  $u$  at each time  $t$ , facilitating the enforcement of display constraint. Second, we also keep track of the set of users to whom item  $i$  has been recommended so far, facilitating the enforcement of capacity constraint. Third, for each user  $u$  and each item class  $c$ , we book-keep the set of triples in  $S$  involving  $u$  and items of class  $c$ . That is,  $\text{set}_{u,c} := \{(u, i, t) \in S \mid \mathcal{C}(i) = c, t \in [T]\}$ . This is needed for marginal revenue computation and lazy forward.

**Two-Level Heaps Data Structure.** For each user-item pair  $(u, i)$  with a positive primitive adoption probability for some time step, we create a priority queue (implemented as a maximum binary heap) to store the marginal revenue of all triples  $(u, i, *)$ , where  $*$  denotes any applicable time steps (line 4). Such heaps are initially populated with the revenue of all triples computed using primitive

---

#### Algorithm 1: G-Greedy (Two-Level Heaps & Lazy Forward)

---

**Input** :  $U, I, T, k, \{q_i\}, \{p(i, t)\}, \{q(u, i, t)\}, \{\beta_i\}$ .  
**Output**: A valid strategy  $S \subseteq U \times I \times [T]$ .

```

1  $S \leftarrow \emptyset$ ; /* initialization */;
2  $\text{upper\_heap} \leftarrow$  an empty maximum binary heap;
3 foreach  $(u, i) \in U \times I$  such that  $\exists q(u, i, t) > 0$  do
4    $\text{lower\_heap}_{u,i} \leftarrow$  an empty maximum binary heap;
5 foreach  $u \in U$ , item class  $c$  do
6    $\text{set}_{u,c} \leftarrow \emptyset$ ;
7 foreach  $(u, i, t) \in U \times I \times [T]$  with  $q(u, i, t) > 0$  do
8    $\text{lower\_heap}_{u,i}.\text{Insert}((u, i, t), q(u, i, t) \cdot p(i, t))$ ;
9    $\text{flag}((u, i, t)) \leftarrow 0$ ; /* for lazy forward */;
10  $\text{upper\_heap}.\text{Heapify}()$  /* populate and heapify it with roots of all lower-level heaps */;
11 while  $|S| < k \cdot T \cdot |U| \wedge \text{upper\_heap}$  is not empty do
12    $z \leftarrow \text{upper\_heap}.\text{FindMax}()$ ; /* root */;
13   if  $Rev_S(z) < 0$  then break; /* negative case */;
14   if  $S \cup \{z\}$  doesn't violate any constraint then
15     if  $\text{flag}(z) < |\text{set}_{z.u, \mathcal{C}(z.i)}|$  then
16       foreach triple  $z' \in \text{lower\_heap}_{z.u, z.i}$  do
17         calculate  $Rev_S(z')$ ; /* Eq. (4) */;
18          $\text{flag}(z') \leftarrow |\text{set}_{z.u, \mathcal{C}(z.i)}|$ ;
19         update  $\text{lower\_heap}_{z.u, z.i}$  and  $\text{upper\_heap}$ ;
20         /* using Decrease-Key on heaps */;
21       else if  $\text{flag}(z) == |\text{set}_{z.u, \mathcal{C}(z.i)}|$  then
22          $S \leftarrow S \cup \{z\}$ ;
23          $\text{set}(z.u, \mathcal{C}(z.i)).\text{Add}(z)$ ;
24          $\text{upper\_heap}.\text{DeleteMax}()$ ;
25       else /* remove from considerations */
26          $\text{upper\_heap}.\text{DeleteMax}()$ ;
         delete  $\text{lower\_heap}_{z.u, z.i}$ ;

```

---

adoption probabilities (line 8). They form the *lower level* and have no direct involvement in seed selection. Then, note that the best candidate triple at any point of the execution must be the root of one of those heaps. Precisely, it is the one with largest marginal revenue amongst all roots. Hence, we sort the roots of all lower-level heaps in a master, *upper-level* priority queue, which is directly involved in seed selection (line 10). Ties are broken arbitrarily.

The intuition is that if we were to use one “giant” heap that contains all triples, it would incur larger overhead in heap operations like Decrease-Key, or Delete-Max, as updated keys will have to traverse a taller binary tree. Conversely, in the two-level structure, each low-level heap contains at most  $T$  elements, and thus the overhead will be almost negligible as long as  $T$  is reasonable (7 in our experiments), while the upper-level heap has at most  $|U| \cdot |I|$  elements, a factor of  $T$  smaller than the “giant one”.

**Lazy Forward and Greedy Selection Details.** By model definition, observe that after a triple is added to  $S$ , the marginal revenue of all triples with the same user and same class of items should be updated before they can be considered for selection. For each update made, a *Decrease-Key* operation is needed in both the lower-level and upper-level heap to reflect the change. However, if a triple’s marginal revenue is small, chances are it will never be percolated up to the root of the upper-level heap. For such triples, an eager update will be wasted and result in inefficiency. Thanks to submodularity of the revenue function (Theorem 2), it is possible to avoid unnecessary computation by using the *lazy forward* optimization proposed in [22] and recently used in [20].

More specifically, we associate with each triple  $z$  a flag variable,

$\text{flag}(z)$ , initialized as 0 (line 9). When a triple  $z = (u, i, t)$  is percolated up to the root of the upper-level heap, we first examine if adding it to  $S$  will violate any constraint. If no violation is caused, and  $\text{flag}(z) = |\text{set}(u, \mathcal{C}(i))|$  holds, then  $z$ 's marginal revenue is *up-to-date* and will be added to  $S$  (lines 20 to 23). If, however,  $\text{flag}(z) < |\text{set}(u, \mathcal{C}(i))|$ , then we retrieve the corresponding lower-level heap, re-compute all stale triples  $\notin S$ , and insert the updated root back to the upper-level heap (lines 15 to 19).

The soundness of lazy forward in G-Greedy stems from submodularity. More specifically, if the root's flag is up-to-date, then its marginal revenue is indisputably the highest regardless of if others are up-to-date (consider any triple ranked lower that is not up-to-date, its actual marginal revenue can be no more than the stale value, due to submodularity). A similar idea was used in the greedy algorithms proposed in [20, 22], where flags are compared to the overall solution size  $|S|$ . But in our case, the revenues from different user-class pairs do not interfere, so we need to check flag values against the size of the corresponding  $\text{set}$  (cf. lines 15 and 20).

**Termination.** The algorithm terminates when one of the following conditions is met: (i). The upper-level heap is exhausted (empty); (ii). All users have received  $k$  recommendations in all time steps; (iii). None of the remaining triples in upper-level heap has a positive marginal revenue. Regardless of the sequence in which the output strategy  $S$  is formed, the final recommendation results are presented to users in natural chronological order.

**Space and Time Complexity.** The upper-level heap has at most  $|U| \times |I|$  triples, while the lower-level heaps, between them, have at most  $|X| = |U| \times |I| \times T$ . Thus total space complexity is  $O(|X|)$ . However, users are typically interested merely in a small subset of items, and thus the *actual triples in consideration can be much fewer than*  $|U| \times |I| \times T$ . For time complexity, it is difficult to analytically estimate how many calls to marginal revenue re-computations are needed with lazy forward, thus we give a *worst-case upper bound* using the number of calls *would have been made without lazy forward*. Let  $y = kT|U|$  be the total number of selections made and the total time complexity is thus  $O(y(|I|T \log T + |I| \log(|U| \times |I|)))$ . This expression is an upper bound corresponding to the worst case scenario where lazy forward is not used (as it is difficult to reason about its actual savings) and all items are in one class (that is, after each addition to  $S$ , all  $|I|$  lower level heaps associated with  $u$  need to be updated). That said, we expect G-Greedy to run much faster in practice because of lazy-forward (a 700 times speedup was reported in [20]).

## 5.2 Two Local Greedy Algorithms

G-Greedy evaluates all candidate triples together and determines a recommendation strategy in a holistic manner. It is interesting to investigate more lightweight heuristics hopefully leading to similar performance. We propose two “local” greedy algorithms that generate recommendations on a *per-time-step* basis. Unlike G-Greedy, these algorithms first finalize  $k$  recommendations to all users for a *single* time step  $t$ , before moving on to another time step  $t'$  until recommendations for all  $T$  time steps are rolled out.

**Sequential Local Greedy (SL-Greedy).** As suggested by its name, this algorithm (presented in Algorithm 2) follows the natural chronological order  $t = 1, 2, \dots, T$  to form recommendations. Note that the key data structures such as  $S$  and  $\text{set}_{u,c}$  are still maintained as global variables for correct computation of marginal revenue. The outer-loop iterates  $T$  times, each corresponding to one time step, and a priority queue (maximum binary heap) is used to sort and store marginal revenue values.

In SL-Greedy, in each iteration  $t$ , the heap only needs to store triples of  $t$  (thus the two-level heaps in G-Greedy are not necessary

---

### Algorithm 2: SL-Greedy

---

**Input** :  $U, I, T, k, \{q_i\}, \{p(i, t)\}, \{q(u, i, t)\}, \{\beta_i\}$ .  
**Output**: A valid strategy  $S \subseteq U \times I \times [T]$ .

```

1  $S \leftarrow \emptyset;$  /* initialization */;
2 foreach  $u \in U$ , item class  $c$  do
3    $\text{set}_{u,c} \leftarrow \emptyset;$ 
4 for  $t = 1$  to  $T$  do
5    $\text{heap} \leftarrow$  an empty maximum binary heap;
6   foreach  $(u, i, t) \in U \times I \times [T]$  do
7     compute  $q_S(u, i, t);$  /* Eq. (2) */
7      $\text{heap.Insert}((u, i, t), p(i, t) \times q_S(u, i, t));$ 
8   while  $\text{heap}$  is not empty do
9      $z \leftarrow \text{heap.FindMax}();$  /* root of heap */;
10    if  $\text{Rev}_S(z) \leq 0$  then break;
11    if  $S \cup \{z\}$  does not violate either constraint then
12       $S \leftarrow S \cup \{z\};$ 
13       $\text{set}(z.u, \mathcal{C}(z.i)).\text{Add}(z);$ 
14      Compute  $\text{Rev}_{S \cup \{z\}}(z.u, j, t), \mathcal{C}(j) = \mathcal{C}(z.i);$ 
15     $\text{heap.DeleteMax}();$ 
```

---

here) and is initially populated with marginal revenue values computed using dynamic adoption probability given  $S$ , which already contains recommended triples up to  $t - 1$  (line 7). The selection procedure is done in a similar fashion to that in G-Greedy, and lazy forward can be applied within each round (i.e., each single time step, lines 5-15 in Algorithm 2). For lack of space, the detailed operations of lazy forward (cf. Algorithm 1) is omitted. SL-Greedy takes  $O(|U| \times |I|)$  space and  $O(y|I| \log(|U| \times |I|))$  time (same reasoning has been applied here as in the case of G-Greedy).

**Randomized Local Greedy.** The natural time ordering used in SL-Greedy (from 1 to  $T$ ) may not be optimal in terms of revenue achieved. To illustrate it, we revisit the example used in the proof of Theorem 2, reproduced below for completeness.

**EXAMPLE 4 (CHRONOLOGICAL IS NOT OPTIMAL).** Let  $U = \{u\}$ ,  $I = \{i\}$ ,  $T = 2$ ,  $k = 1$ ,  $q_i = 2$ ,  $q(u, i, 1) = 0.5$ ,  $q(u, i, 2) = 0.6$ ,  $p(i, 1) = 1$ ,  $p(i, 2) = 0.95$ ,  $\beta_i = 0.1$ . SL-Greedy follows the chronological order  $\langle 1, 2 \rangle$  and outputs a strategy  $S = \{(u, i, 1), (u, i, 2)\}$  with  $\text{Rev}(S) = 0.5285$ . However, if the order of  $\langle 2, 1 \rangle$  were followed, we would have a better strategy  $S' = \{(u, i, 2)\}$  with  $\text{Rev}(S') = 0.57$ . This is because the marginal revenue of  $(u, i, 1)$  w.r.t.  $S'$  is negative and hence will not be added to  $S'$ .

Ideally, we shall determine an optimal permutation of  $[T]$  such that the recommendations generated in that ordering yields the best revenue. However, there seems to be no natural structure which we can exploit to avoid enumerating all  $T!$  permutations for finding the optimal permutation. To circumvent this, we propose RL-Greedy that first repeatedly samples  $N \ll T!$  distinct permutations of  $[T]$  and executes greedy selection for each permutation. Let  $S_j$  be the strategy set chosen by the  $j$ -th execution. In the end, RL-Greedy returns the one that yields the largest revenue, i.e.,  $S = \arg \max_{j=1 \dots N} \text{Rev}(S_j)$ . Under any permutation, the seed selection is done on a per-time-step basis, following lines 5-15 in Algorithm 2. Due to lack of space and similarity to SL-Greedy, we omit the pseudo-code of RL-Greedy. RL-Greedy is a factor of  $N$  slower than SL-Greedy due to repeated sampling and has the same space complexity as SL-Greedy.

## 6. EMPIRICAL EVALUATION



	Amazon	Epinions	Synthetic
#Users	23.0K	21.3K	100K – 500K
#Items	4.2K	1.1K	20K
#Ratings	681K	32.9K	N/A
#Triples with positive $q$	<b>16.1M</b>	<b>14.9M</b>	<b>50M – 250M</b>
#Item classes	94	43	500
Largest class size	1081	52	60
Smallest class size	2	10	24
Median class size	12	27	40

**Table 1: Data Statistics (K: thousand)**

We conduct extensive experiments on two real datasets – Amazon and Epinions (<http://www.epinions.com/>) – to evaluate REVMAX algorithms. Statistics of the datasets are in Table 1.

In the experiments, an important task is to pre-compute the primitive adoption probabilities for each applicable  $(u, i, t)$  triple. The intuition is that if the user is predicted to like the item a lot, i.e., if the predicted rating  $\hat{r}_{ui}$  from an established classical RS is high, then  $q(u, i, t)$  should be high. Intuitively,  $q(u, i, t)$  should be anti-monotone w.r.t. price<sup>1</sup>, and we use the notion of buyer valuation to instantiate this intuition. Let  $val_{ui}$  be user  $u$ ’s valuation on item  $i$ , which is the maximum amount of money  $u$  is willing to pay for getting  $i$ . For trust or privacy reasons users typically do not reveal their true valuations, thus we make the independent value (IPV) assumption which says that  $val_{ui}$  is drawn from a common probability distribution and is independent of others [24, 28]. Thus, we use a simple definition to estimate  $q(u, i, t)$  to be  $\Pr[val_{ui} \geq p(i, t)] \cdot \hat{r}_{ui}/r_{\max}$ , where  $r_{\max}$  is the maximum rating allowed by the system.

For real data, the first step is to compute predicted ratings using a “vanilla” MF model (we used the stochastic gradient descent algorithm) [18]. Then, for all users we select 100 items with the highest predicted ratings and compute primitive adoption probabilities (if the rating is too low, the item is deemed to be of little interest). Naturally, only triples with nonzero adoption probability will be considered by (and given as input to) any REVMAX algorithm, and thus *the number of such triples is the true input size* (as an analogy, recall that the number of known ratings is the true input size that affects the running time of RS algorithms such as matrix factorization). We highlight this number in Table 1 in bold font.

## 6.1 Data Preparations and Experiments Setup

**Amazon.** We selected 5000 popular items from the Electronics category and crawled their prices from August 31, 2013 to November 1, 2013 via Amazon’s Product Advertising API<sup>2</sup>. The reason to select popular items is that they receive enough ratings for computing predicted ratings in a reliable manner. The reason to focus on one category is that we want the buyers of different items to have a reasonable overlap. The items include, e.g., Amazon Kindle and accessories, Microsoft Xbox 360 and popular Xbox games, etc. For all items, we record one price per day. In addition, we gathered all historical ratings of these items and the users providing them. Items with fewer than 10 ratings are filtered out. We then train a low-rank matrix model using the implementation in MyMediaLite [12] to obtain predicted ratings. The model yields a RMSE of 0.91 on five-

fold cross validation, which is reasonably good by RS standards. We also set  $T = 7$  to simulate a horizon of one week.

**Epinions.** When giving reviews, Epinions users can optionally report the price they paid (in US dollars). This makes Epinions a valuable source for obtaining price data. We extracted item information from a public Epinions dataset [27] and followed the provided URLs (of the product pages) to crawl all reviews and prices. For accurate estimations of price and valuation distributions, items having fewer than 10 reported prices were filtered out. We also trained a matrix factorization model with a RMSE of 1.04 on five-fold cross validation. It has been noted before that Epinions is an ultra sparse dataset and hence has a higher RMSE [12].

**Learning Price and Valuation Distributions.** The prices on Epinions cannot be mapped to a ground-truth time-series as users bought the item from many different sellers. To circumvent this, we apply the *kernel density estimation* (KDE) method [29] to estimate price distributions. Consider an arbitrary item  $i$  and let  $\langle p_1, p_2, \dots, p_{n_i} \rangle$  be the list of prices reported for  $i$ . In KDE, it is assumed that the  $n_i$  prices are i.i.d. with a probability distribution whose density function takes the form  $\hat{f}_i(x) = \frac{1}{n_i h} \sum_{j=1}^{n_i} \kappa\left(\frac{x-p_j}{h}\right)$ , where  $\kappa(\cdot)$  is the kernel function and  $h > 0$  is a parameter called *bandwidth*, controlling the scale of smoothing. We apply the Gaussian kernel [14] by setting  $\kappa(x)$  to be the standard Gaussian density function  $\phi(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})$ . The optimal bandwidth  $h$  for the Gaussian kernel can be determined by Silverman’s rule of thumb [29]:  $h^* = (\frac{4\hat{\sigma}^5}{3n_i})^{\frac{1}{5}}$ , where  $\hat{\sigma}$  is the empirical standard deviation. Then, from the estimated  $\hat{f}_i$ , we generate  $T = 7$  samples and treat the samples as if they were the prices of  $i$  in a week. For each item  $i$  we also use  $\hat{f}_i$  as a proxy for its valuation distribution. Note that the distribution  $\hat{f}_i$  remains Gaussian with mean  $\mu_i = \sum_{j=1}^{n_i} p_j / (n_i h)$  and variance  $\sigma_i^2 = h$ . Thus for any price value  $p$ ,  $\Pr[val_{ui} \geq p] = 1 - \hat{F}_i(p) = \frac{1}{2}(1 - \text{erf}(\frac{p-\mu_i}{\sqrt{2\sigma_i}}))$ , where  $\text{erf}(\cdot)$  is the Gauss error function.

**Synthetic Data.** We use synthetic datasets much larger than Amazon and Epinions to gauge the scalability of algorithms. We do *not* report revenue achieved on this data, since the generation process is artificial and *this dataset is solely used for testing scalability*. In total there are five datasets with  $|U| = 100K, 200K, \dots, 500K$ .  $T$  is set to 5. The item-set is the same:  $|I| = 20K$  and for each item  $i$ , choose a value  $x_i$  uniformly at random from  $[10, 500]$ , and for each time  $t$ , sample  $p(i, t)$  uniformly at random from  $[x_i, 2x_i]$ . For each user  $u$ , we randomly choose 100 items to be the highest rated, and for each such item, sample  $T$  (primitive) adoption probability values from a Gaussian distribution with  $\mu = y_i$  and  $\sigma^2 = 0.1$ , where  $y_i$  itself is chosen randomly from  $[0, 1]$ ; then we match adoption probabilities with prices so that anti-monotonicity holds. Input size will then be  $100T|U|$  (cf. Table 1). Recall, the number of nonzero adoption probabilities is the critical factor in deciding the scalability of any REVMAX algorithm.

**Parameter Settings for Amazon and Epinions.** It is unrealistic and unlikely that a user would be interested in buying all items. That is, for each user  $u$ ,  $q(u, i, t)$  will be nonzero for a small subset of items: we rank the items based on their predicted ratings for  $u$  and compute adoption probabilities for the top-100 items. We also need the saturation factor  $\beta_i$  for computing marginal revenue. We test the following two cases. First, we hard-wire a uniform value for all items, testing three different cases: 0.1, 0.5, and 0.9, representing strong, medium, and weak effect of saturation. Second, for each item  $i \in I$ , its  $\beta_i$  is chosen uniformly at random from  $[0, 1]$  – this effectively averages over possible values for all the items to

<sup>1</sup>Our framework and algorithms do not assume this.

<sup>2</sup><https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html>

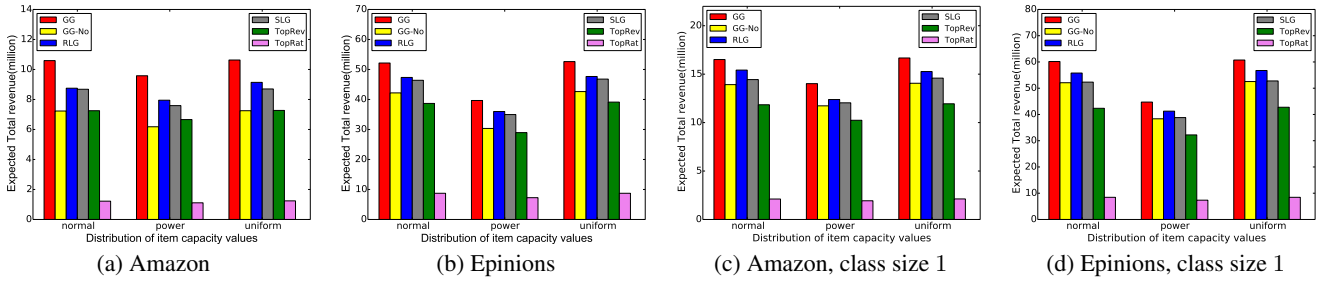


Figure 1: Expected total revenue achieved, with each  $\beta_i$  chosen uniformly at random from  $[0, 1]$ .

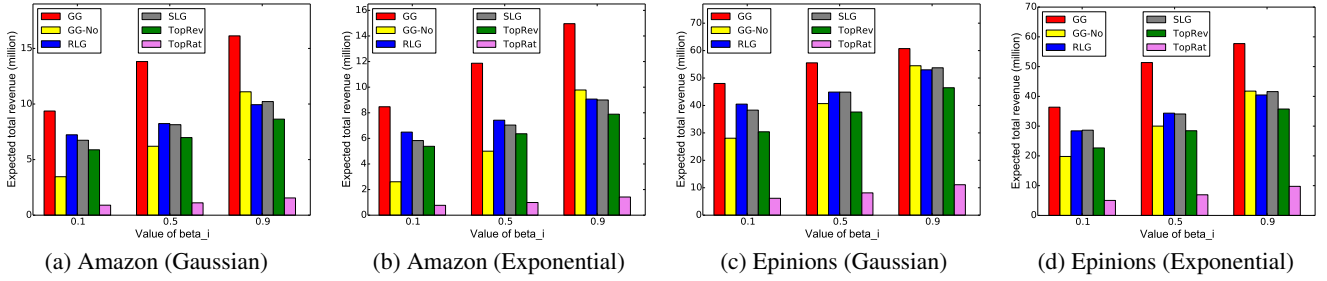


Figure 2: Expected total revenue with varying saturation strength, item class size  $> 1$ .

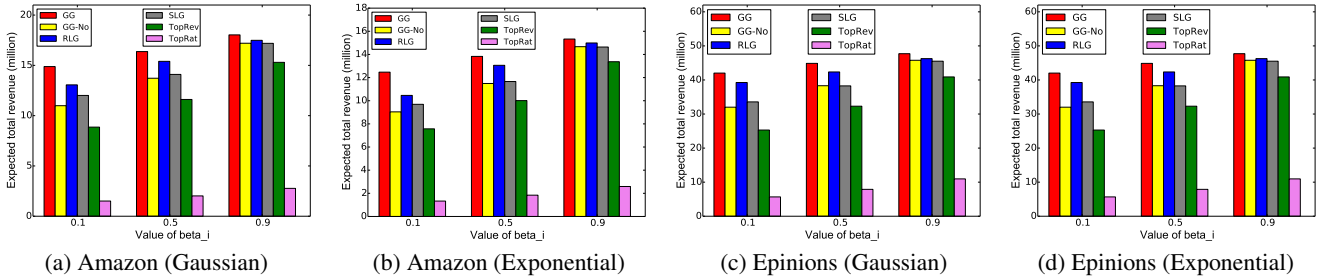


Figure 3: Expected total revenue with varying saturation strength, item class size  $= 1$ .

model our lack of knowledge. For capacity constraints  $q_i$ , we consider two probability distributions from which  $q_i$  is sampled: (1) Gaussian:  $\mathcal{N}(5000, 200)$  for Epinions and  $\mathcal{N}(5000, 300)$  for Amazon; (2) exponential with inverse scale  $2 \times 10^{-3}$  (mean 5000). We believe the above scenarios are representative and worth testing.

**Algorithms Evaluated.** We compare G-Greedy, SL-Greedy, and RL-Greedy with the following natural baselines. TopRA (for Top Rating) recommends to every user the  $k$  items with highest predicted rating by MF; TopRE (for Top REvenue) recommends to every user the  $k$  items with highest “expected revenue” (price  $\times$  primitive adoption probability). Since TopRA is inherently “static”, to evaluate the expected revenue it yields over  $[T]$ , the recommended items are repeated in all  $T$  time steps. We also consider a “degenerated” version of G-Greedy, which we call GlobalNo: it ignores saturation effects when selecting triples. That is, when computing marginal revenue and selecting triples, GlobalNo will behave as though  $\beta_i = 1$  for all  $i \in I$ , but when we compute the final revenue yielded by its output strategy, the true  $\beta_i$  values will be used. This is to measure how much revenue would be lost if saturation effects are present but ignored.

## 6.2 Results and Analysis

Experimental results are reported on two metrics: expected revenue achieved (quality of recommendations, *cf.* Def. 2) and number

of repeated recommendations made. In addition to using ground-truth information on item classes from the data, we also test the scenario with class size  $= 1$ , i.e., each item is in its own category. For RL-Greedy, we generate  $N = 20$  permutations of  $[T]$ . All implementations are in Java and programs were run on a Linux server with Intel Xeon CPU X5570 (2.93GHz) and 256GB RAM.

**Quality of Recommendations.** Figure 1 shows the expected total revenue achieved by various algorithms and baselines when  $\beta_i$  is chosen uniformly at random from  $[0, 1]$ . As can be seen, G-Greedy *consistently* yields the best revenue, leading the runner-up RL-Greedy by non-trivial margins (about 10% to 20% gain). GlobalNo is always behind G-Greedy (about 10% to 30% loss), and so is SL-Greedy compared to RL-Greedy (about 1% to 6% behind). TopRE and TopRA are always outperformed by all greedy algorithms (though it is less so for TopRE). G-Greedy is typically 30% to 50% better than TopRE. Note that the expected revenues are in the scale of tens of millions dollars, so even a small relative gain could translate to large revenue.

Figure 2 and Figure 3 show the comparisons of revenue with uniform  $\beta_i$  values: 0.1, 0.5, and 0.9. The purpose is to examine how algorithms “react” to different strength of saturation effects. As can be seen, the hierarchy of algorithms (ranked by revenue) is quite consistent with that in Figure 1, and importantly G-Greedy is always the top performer. The gap between G-Greedy and the rest

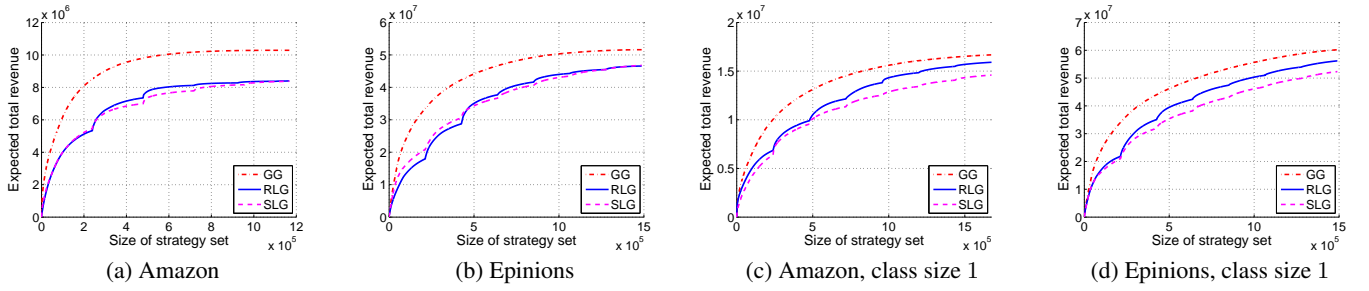


Figure 4: Expected total revenue of G-Greedy, SL-Greedy, RL-Greedy vs. solution size ( $|S|$ ).

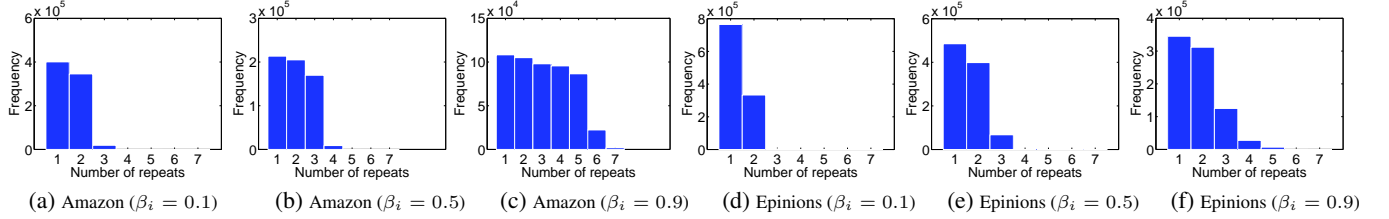


Figure 5: Histogram on the number of repeated recommendations made by G-Greedy for each user-item pair.

	GG	RLG	SLG	TopRE	TopRA
Amazon	4.67	6.81	7.95	0.78	0.45
Epinions	2.35	3.00	2.71	0.68	0.16

Table 2: Running time (in mins) comparison

is larger with smaller  $\beta_i$  (stronger saturation). In Figure 3 (class size 1), though SL-Greedy is always behind RL-Greedy, the difference becomes smaller as  $\beta_i$  increases. This intuitively suggests that RL-Greedy makes better decisions when it comes to repeated recommendations, as it is less sensitive to strong saturation.

In Figure 4, we plot the growth of revenue as the greedy algorithms increment  $S$  (Gaussian item quantities,  $\beta_i$  uniform in  $[0, 1]$ ). The lines for G-Greedy clearly illustrate the phenomenon of diminishing marginal returns, empirically illustrating submodularity. Interestingly enough, SL-Greedy and RL-Greedy have similar overall trends but also have “segments”, corresponding to switches in time steps; submodularity can be observed within each “segment”.

Finally, Figure 5 presents histograms on the number of repeated recommendations made by G-Greedy for each user-item pair (item class size  $> 1$ ; other cases are similar and hence omitted). The Y-axis is normalized to show the percentage instead of absolute counts. Note that in both datasets, when  $\beta_i = 0.1$ , it happens more often that an item is recommended only once or twice to a user, since the dynamic adoption probability drops rapidly with small  $\beta_i$ . As  $\beta_i$  becomes large, the histogram becomes less skewed as more repeats show up, especially on Amazon with  $\beta_i = 0.9$ . This experiment shows that G-Greedy takes advantage of (lack of) saturation for making repeat recommendations to boost revenue.

**Running Time and Scalability Tests.** Table 2 reports the running time of various algorithms on Amazon and Epinions. For lack of space, we only show the numbers for cases with uniform random  $\beta_i$  and Gaussian item capacities (other cases are similar and hence omitted). On both datasets, G-Greedy, SL-Greedy and RL-Greedy show scalability and efficiency and all finish under 10 minutes for Amazon and under 5 minutes for Epinions.

In addition, we run G-Greedy on synthetic datasets that are much larger than Amazon and Epinions. Figure 6 illustrates the running

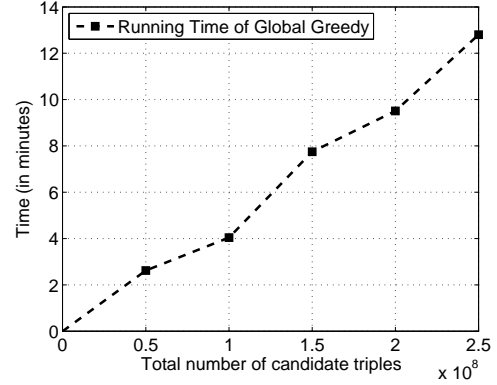


Figure 6: Running time of G-Greedy on synthetic data

time of G-Greedy on synthetic data with 100K, 200K, 300K, 400K, and 500K users with  $T = 5$  and each user having 100 items with non-zero adoption probability. This means, for example, the dataset with 500K users has 250 million triples to select from. The growth rate in Figure 6 is almost linear and it takes about 13 minutes to finish on the largest one, which clearly demonstrates the scalability of G-Greedy. To put things in perspective, Netflix, the largest public ratings dataset, has only 100 million known ratings.

In sum, our experiments on Amazon and Epinions data show that the proposed greedy algorithms are effective and efficient, producing far superior solutions than the baselines. In particular, the most sophisticated G-Greedy consistently outperforms the rest and easily scales to (synthetic) data 2.5 times the size of the Netflix dataset.

Next, in the following subsection, we report additional experimental results conducted in the settings where information about product prices is not completely available at the beginning of a time horizon.

### 6.3 Experimental Results: Incomplete Product Prices

So far, our experiments have focused on the setting where all product prices, i.e.,  $p(i, t)$ , for all  $t$  in time horizon  $[T]$ , are avail-



able as input to algorithms like G-Greedy and RL-Greedy when they are making recommendation decisions. However, this may not always be the case in practice, as prices are dynamic and exact values may not be available much in advance. Notice that this does not affect SL-Greedy as it only requires prices for the current time. As such, we are interested in gauging how our algorithms perform in the setting where product prices become available in batches and in time order.

More specifically, the time horizon  $[T]$  is “divided” into sub-horizons  $[T_1], [T_2], \dots, [T_r]$ , and prices become available sub-horizon after sub-horizon. For example, suppose  $T = 7$ ,  $[T_1] = \{1, 2, 3\}$ , and  $[T_2] = \{4, 5, 6, 7\}$ . Initially, only prices for time steps 1 through 3 are known to the recommender system, and only at  $t = 4$ , prices for time steps 4 through 7 will be known. To adapt G-Greedy and RL-Greedy, the algorithms will first come up with recommendations for  $[T_1]$ , and then given those, consider recommendations in  $[T_2]$ . We expect both G-Greedy and RL-Greedy to yield less revenue under this setting because they are no longer able to select triples in a holistic manner for the entire horizon  $[T]$ . Also, note that SL-Greedy is not affected at all since it already makes recommendations in chronological order.

In our experiments, we set  $T = 7$  and split it into two sub-horizons, with cut-off time at 2, 4, and 5, respectively. For example, if cut-off is 2, then  $[T_1] = \{1, 2\}$  and  $[T_2] = \{3, 4, 5, 6, 7\}$ . Figure 7 shows the revenue achieved by G-Greedy and RL-Greedy w.r.t. the three different cut-off time steps, and compares with the revenue yielded by these algorithms when prices are available all at once (hereafter referred to as the “original setting”). Both Amazon and Epinions datasets are used. As can be seen from Figure 7, G-Greedy (GG<sub>2</sub>, GG<sub>4</sub>, and GG<sub>5</sub>) still outperforms RL-Greedy (RLG<sub>2</sub>, RLG<sub>4</sub>, and RLG<sub>5</sub>) and SL-Greedy. In the case of G-Greedy, GG<sub>2</sub>, GG<sub>4</sub>, and GG<sub>5</sub> are all worse than the original setting (which is not surprising), and the loss is the greatest when the cut-off time is 4, the most even split on  $[T]$ . This may be because when the split is not even (e.g., GG<sub>2</sub> and GG<sub>5</sub>), the algorithm gets to see a larger bulk of the price information together compared with the even split (GG<sub>4</sub>). Similar trends can be observed for RL-Greedy.

## 7. EXTENSION TO RANDOM PRICES

So far, we have assumed that we have access to an exact pricing model, whereby the exact price of items within a short time horizon is known. While there is evidence supporting this assumption and there are microeconomics studies on exact pricing model as pointed out in §1, the question arises what if the price prediction model is probabilistic in nature. More precisely, by treating prices  $p(i, t)$  as random variables, such a model only predicts them to within a distribution. Can we leverage the theory and techniques developed in this paper to deal with this case? We settle this question in this section. It is easy to see REVMAX remains NP-hard, by restricting to the case where all prices are known exactly w.p. 1. The interesting question is how the algorithms can be leveraged for the random price model. In this case, the expected total revenue yielded by a strategy would be an expectation taken over the randomness of adoption events (as in the exact-pricing model) and over the randomness of prices. A major complication is that a closed form for the revenue function may not exist depending on the distribution from which prices are drawn. An obvious way of dealing with this is to treat the expected price (or most probable price) as a proxy for exact price and find a strategy that optimizes the expected revenue w.r.t. this assumption, using algorithms in §5. It is a heuristic and will clearly be suboptimal w.r.t. the true expected revenue. A more principled approach is to appeal to the Taylor approximation

method used in convergence analysis [5]. It has the advantage of being distribution independent.

Consider any valid strategy  $S$ , and a triple  $z = (u, i, t) \in S$ . Define  $[z]_S := \{(u, j, t') \in S : \mathcal{C}(j) = \mathcal{C}(i) \wedge t' \leq t\}$ . That is,  $[z]_S$  contains the triples that “compete” with  $z$  under  $S$ . Now consider the price random variables corresponding to all triples in  $[z]_S$ . E.g., if items  $i_1, i_2, i_3$  are from the same class,  $S = \{(u, i_1, t_1), (u, i_2, t_2), (u, i_3, t_3)\}$ ,  $t_1 < t_2 < t_3$ , and  $z = (u, i_3, t_3)$ , then the revenue contribution of  $z$  to  $S$  will be dependent on the price vector  $(p(i_1, t_1), p(i_2, t_2), p(i_3, t_3))$ . For notational simplicity, we let  $\mathbf{z}$  be this price vector for  $z$ , and use  $\mathbf{z}_a$  to denote the  $a$ -th coordinate of  $\mathbf{z}$ . In the above example, if  $a = 2$ , then  $\mathbf{z}_a = p(i_2, t_2)$ . The contribution from triple  $z$  to the overall revenue of  $S$  is clearly a function of the price vector  $\mathbf{z}$ , and we denote it by  $g(\mathbf{z})$ . For all  $a = 1, 2, \dots, |\mathbf{z}|$ , let  $\bar{\mathbf{z}}_a$  and  $\text{var}(\mathbf{z}_a)$  be the mean and variance of  $\mathbf{z}_a$  respectively. Also, let  $\text{cov}(\mathbf{z}_a, \mathbf{z}_b)$  be the covariance of two prices  $\mathbf{z}_a$  and  $\mathbf{z}_b$ , where  $a \neq b$ . Finally, by  $\bar{\mathbf{z}}$ , we mean the vector of means of the price random variables in  $\mathbf{z}$ . In our running example,  $\bar{\mathbf{z}} = (\bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2, \bar{\mathbf{z}}_3)$ , where  $\mathbf{z}_1 = p(i_1, t_1)$ , etc. We then expand  $g(\mathbf{z})$  at  $\bar{\mathbf{z}}$  using *Taylor’s Theorem*:

$$g(\mathbf{z}) = g(\bar{\mathbf{z}}) + \sum_{a=1}^{|\mathbf{z}|} \frac{\partial g(\bar{\mathbf{z}})}{\partial \mathbf{z}_a} (\mathbf{z}_a - \bar{\mathbf{z}}_a) + \frac{1}{2} \sum_{a=1}^{|\mathbf{z}|} \sum_{b=1}^{|\mathbf{z}|} \frac{\partial^2 g(\bar{\mathbf{z}})}{\partial \mathbf{z}_a \partial \mathbf{z}_b} (\mathbf{z}_a - \bar{\mathbf{z}}_a)(\mathbf{z}_b - \bar{\mathbf{z}}_b) + r(\mathbf{z}), \quad (7)$$

where  $r(\mathbf{z})$  is the remainder (consisting of higher order terms) and by Taylor’s theorem it satisfies  $\lim_{\mathbf{z} \rightarrow \bar{\mathbf{z}}} \frac{r(\mathbf{z})}{(\mathbf{z} - \bar{\mathbf{z}})^2} = 0$ . Following standard practice [5], this remainder is ignored since we are interested in an efficient approximation.

Disregarding the remainder and taking expectation over both sides of (7) gives the expected revenue contribution of triple  $z$ :

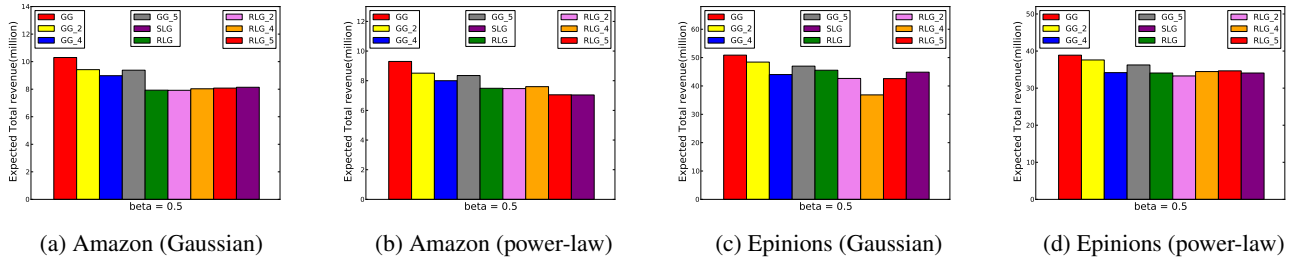
$$\begin{aligned} \mathbb{E}[g(\mathbf{z})] &\approx \mathbb{E}[g(\bar{\mathbf{z}})] + \sum_{a=1}^{|\mathbf{z}|} \frac{\partial g(\bar{\mathbf{z}})}{\partial \mathbf{z}_a} \mathbb{E}[(\mathbf{z}_a - \bar{\mathbf{z}}_a)] + \\ &\frac{1}{2} \sum_{a=1}^{|\mathbf{z}|} \sum_{b=1}^{|\mathbf{z}|} \frac{\partial^2 g(\bar{\mathbf{z}})}{\partial \mathbf{z}_a \partial \mathbf{z}_b} \mathbb{E}[(\mathbf{z}_a - \bar{\mathbf{z}}_a)(\mathbf{z}_b - \bar{\mathbf{z}}_b)] \\ &= g(\bar{\mathbf{z}}) + \frac{1}{2} \sum_{a=1}^{|\mathbf{z}|} \text{var}(\mathbf{z}_a) + \sum_{1 \leq a < b \leq |\mathbf{z}|} \text{cov}(\mathbf{z}_a, \mathbf{z}_b), \quad (8) \end{aligned}$$

where we have applied the linearity of expectation to get  $\mathbb{E}[\mathbf{z}_a - \bar{\mathbf{z}}_a] = 0, \forall a$ . Thus, for any strategy  $S$ , its expected total revenue, denoted  $\text{RandRev}(S)$ , is  $\text{RandRev}(S) = \sum_{z \in S} \mathbb{E}[g(\mathbf{z})]$ .

The first three summands in (8) correspond to mean (first-order), variance, and covariance (second-order) respectively. They are used to estimate the true revenue function and the remainder is ignored. More precisely, the algorithms for the exact-price model can be used, with the calculation of revenue changed by adding the extra variance and covariance terms as shown in (8). In principle, we can incorporate as many terms from the Taylor expansion as dictated by the accuracy desired and simply use the algorithms in §5 for finding strategies with large revenue.

## 8. CONCLUSIONS AND FUTURE WORK

In this work, we investigate the business-centric perspective of RS, and propose a dynamic revenue model by incorporating many crucial aspects such as price, competition, constraints, and saturation effects. Under this framework, we study a novel problem REVMAX, which asks to find a recommendation strategy that max-



**Figure 7: Revenue comparison for complete prices with gradual availability. In the legend,  $GG_i$  and  $RLG_i$  means that the first sub-horizon is from time step 1 to  $i$ , and the second sub-horizon is from time step  $i + 1$  to  $T$ .**

imizes the expected total revenue over a given time horizon. We prove that REVMAX is NP-hard and develop an approximation algorithm for a slightly relaxed version ( $R$ -REVMAX) by establishing an elegant connection to matroid theory. We also design intelligent greedy algorithms to tackle the original REVMAX and conduct extensive experiments on Amazon and Epinion data to show the effectiveness and efficiency of these algorithms. Furthermore, using synthetic data, we show that the G-Greedy algorithm scales to datasets 2.5 times the size of the Netflix dataset.

For future work, on the theoretical side, it is worth asking if REVMAX remains NP-hard when every item belongs to its own class. For the random price model, it is interesting to investigate if the Taylor approximation method can yield a strategy with a guaranteed approximation to the optimal solution w.r.t. true expected revenue. In reality, it is possible that prices, saturation, and competition may interact. Modeling and learning the interactions present is an interesting challenge. On the practical side, an interesting challenge is to find suitable real datasets from which to learn the parameters for the random price model and conduct empirical evaluations on it. Finally, here we have focused on revenue-maximizing recommendation problem, given an exogenous price model. Conversely, to find optimal pricing in order to maximize the expected revenue in the context of a given RS is an interesting problem which has clear connections to algorithmic game theory [24, 28].

## 9. REFERENCES

- [1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.
- [2] J. Angwin and D. Mattioli. Coming soon: Toilet paper priced like airline tickets. *Wall Street Journal*, September 5, 2012, <http://on.wsj.com/1lECov1>
- [3] A. Azaria et al. Movie recommender system for profit maximization. In *RecSys*, pages 121–128, 2013.
- [4] Camelytics. *Prices Always Change*, 2012 (accessed May 9, 2014). <http://bit.ly/1jznFgL>.
- [5] G. Casella and R. L. Berger. *Statistical Inference (2nd edition)*. Duxbury, 2002.
- [6] L.-S. Chen et al. Developing recommender systems with the consideration of product profitability for sellers. *Inf. Sci.*, 178(4):1032–1048, 2008.
- [7] A. Das, C. Mathieu, and D. Ricketts. Maximizing profit using recommender systems. *CoRR*, abs/0908.3633, 2009.
- [8] A. Das-Sarma et al. Understanding cyclic trends in social choices. In *WSDM*, pages 593–602, 2012.
- [9] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. In *FOCS*, pages 184–193, 1975.
- [10] H. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC*, pages 448–456, 1983.
- [11] H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.
- [12] Z. Gantner et al. Mymedialite: a free recommender system library. In *RecSys*, pages 305–308, 2011.
- [13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [14] A. X. Jiang et al. Bidding agents for online auctions with hidden bids. *Machine Learning*, 67(1-2):117–143, 2007.
- [15] S. Kalish. A new product adoption model with price, advertising, and uncertainty. *Management Science*, 31(12):1569–1585, 1985.
- [16] K. Kapoor et al. Measuring spontaneous devaluations in user preferences. In *KDD*, pages 1061–1069, 2013.
- [17] Y. Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, 2010.
- [18] Y. Koren et al. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.
- [19] J. Lee, V. S. Mirrokni, V. Nagarajan, and M. Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM J. Discrete Math.*, 23(4):2053–2078, 2010.
- [20] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1), 2007.
- [21] F. Manshadi et al. A distributed algorithm for large-scale generalized matching. *PVLDB*, 6(9):613–624, 2013.
- [22] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *IFIP Conf. on Optimization Techniques*, page 234–243, 1978.
- [23] G. D. F. Morales, A. Gionis, and M. Sozio. Social content matching in mapreduce. *PVLDB*, 4(7):460–469, 2011.
- [24] N. Nisan et al. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.
- [25] E. L. Porteus. Stochastic inventory theory. In *Stochastic Models*, volume 2, Elsevier, 1990.
- [26] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.
- [27] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.
- [28] Y. Shoham and K. Leyton-Brown. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*.

Cambridge University Press, 2009.

- [29] B. W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.
- [30] C. Snyder and W. Nicholson. *Microeconomic Theory, Basic Principles and Extensions (10th ed)*. South-Western Cengage Learning, 2008.
- [31] J. Wang and Y. Zhang. Opportunity model for e-commerce recommendation: right product; right time. In *SIGIR*, pages 303–312, 2013.
- [32] G. Zhao, M.-L. Lee, W. Hsu, and W. Chen. Increasing temporal diversity with purchase intervals. In *SIGIR*, pages 165–174, 2012.